Stork User Manual

version 1.0.0

1 Quick Start

```
$ tar xvfz stork_binary_1.0.0.tar.gz
$ cd stork_binary_1.0.0/
$ ls
release.tar.gz  stork-install.ksh
$
$ ./stork-install.ksh
...
$ export STORK_CONFIG=/usr/local/stork/etc/stork_config
$ export PATH:$PATH:/usr/local/stork/bin:/usr/local/stork/sbin
```

Extract Stork binary package and execute the installation script. Set environment variable *STORK_CONFIG* pointing to your configuration file. Start the server by running *stork_server* command. Check sample submit files in *etc/* under Stork installation directory. Prepare your job submission file and submit using *stork_submit*.

2 Installation

First, extract the Stork binary package to get the *release.tar.gz* and the installation script, *stork-install.ksh*.

```
$ tar xvfz stork_binary_1.0.0.tar.gz
  cd stork_binary_1.0.0/
```

In the current directory, run Stork installation script by giving the installation directory as argument (by default it will try to install into */usr/local/stork*).

```
 ./stork-install.ksh ~user1/stork

Installation directory is /home/user1/stork / Proceed (yes/no)?
yes
...
./stork-install.ksh: extracting binaries
                        release_file: /tmp/release.tar.gz
./
./bin/
./bin/stork_rm
./bin/stork_submit
./bin/stork_status
./bin/stork_q
./local/
./local/log/
./.stork_home
./etc/
./etc/stork_config.sample
./etc/submitArgument.sample
./etc/stork_config.template
./etc/stork_config
./etc/submit.sample
./libexec/
./libexec/stork.transfer.castor_srm
./libexec/stork.globus-url-copy
./libexec/stork.transfer.srb
./libexec/stork.transfer.globus-url-copy
./libexec/stork.transfer.dcache_srm
./libexec/stork.transfer.petashare
./libexec/stork.transfer.irods
./libexec/stork.transfer.file-file
./libexec/stork.transfer.unitree
./sbin/
./sbin/stork_server

./stork-install.ksh: creating configuration file

 Please set STORK_CONFIG environment variable!

export STORK_CONFIG=/home/user1/stork/etc/stork_config

By default, stork_server will search /usr/local/stork/etc,
for "stork_config" file if "STORK_CONFIG" environment variable is not defined!

 Please set PATH environment variable to Stork binaries

export PATH=$PATH:/home/user1/stork/bin:/home/user1/stork/sbin

Installation Complete!
```

The *STORK_CONFIG* environment variable should point to the *stork_config* file. Installation script prepares the configuration file and puts under *etc/* in the installation folder. By default, *stork_server* will search */usr/local/stork/etc* for *stork_config* file if *STORK_CONFIG* environment variable is not defined.

In the installation folder, we have the following directory structure;

```
.:
```

```
bin/  etc/  libexec/  local/  sbin/

./bin:
stork_q
stork_rm
stork_status
stork_submit

./etc:
stork_config
stork_config.template
stork_config.sample
submit.sample
submitArgument.sample

./libexec:
 stork.transfer.file-file
 stork.transfer.dcache_srm
 stork.transfer.castor_srm
 stork.transfer.petashare
 stork.transfer.globus-url-copy
 stork.transfer.unitree
 stork.globus-url-copy
 stork.globus-url-copy64
 stork.transfer.irods
 stork.transfer.srb

./local:
log/

./local/log:

./sbin:
stork_server
```

Please note that Stork uses Globus components for file transfers using the *GsiFTP* protocol (GridFTP [1]). Therefore, you need to have Globus installed and configured in your system with user/host certificates. Stork will perform the requested operation using *stork.globus-url-copy* command in the *libexec/* directory. It is basically an extended version (adding some extra features to be used by Stork) of *globus-url-copy* from Globus [2]. You can replace that command by *globus-url-copy* in your system. Simply delete the file and put a link named as *stork.globus-url-copy* pointing to the gridFTP client utility in your system.

We have Stork commands in *bin/* directory, and Stork server in *sbin/*. Sample submit files and templates for Stork configuration can also be found in *etc/* directory. Stork log files are kept in *local/log/* by default. Transfer modules are in *libexec/* and installation script will create appropriate symbolic links for each

[1] http://www.globus.org/grid_software/data/gridftp.php
[2] http://www.globus.org/toolkit/

transfer pair (i.e. *gsiftp* to *file://*).

```
stork.transfer.file-file

stork.transfer.ftp-file -> stork.transfer.globus-url-copy*
stork.transfer.file-ftp -> stork.transfer.globus-url-copy*
stork.transfer.ftp-ftp -> stork.transfer.globus-url-copy*

stork.transfer.gsiftp-file -> stork.transfer.globus-url-copy*
stork.transfer.file-gsiftp -> stork.transfer.globus-url-copy*
stork.transfer.gsiftp-ftp -> stork.transfer.globus-url-copy*
stork.transfer.ftp-gsiftp -> stork.transfer.globus-url-copy*
stork.transfer.gsiftp-gsiftp -> stork.transfer.globus-url-copy*
stork.transfer.http-gsiftp -> stork.transfer.globus-url-copy*

stork.transfer.file-petashare -> stork.transfer.petashare*
stork.transfer.petashare-file -> stork.transfer.petashare*
stork.transfer.petashare-petashare -> stork.transfer.petashare*

stork.transfer.irods-file -> stork.transfer.irods*
stork.transfer.file-irods -> stork.transfer.irods*

stork.transfer.file-srb -> stork.transfer.srb*
stork.transfer.srb-file -> stork.transfer.srb*
```

Default values can be edited inside the *stork_config* file:

```
##########################################################################
##
##  stork_config
##
##########################################################################
RELEASE_DIR             =  /tmp/user1/stork

##########################################################################

LOCAL_DIR               = $(RELEASE_DIR)/local

##########  Pathnames

LOG             = $(LOCAL_DIR)/log
SPOOL           = $(LOCAL_DIR)/spool
EXECUTE         = $(LOCAL_DIR)/execute
BIN             = $(RELEASE_DIR)/bin
LIB             = $(RELEASE_DIR)/lib
INCLUDE         = $(RELEASE_DIR)/include
SBIN            = $(RELEASE_DIR)/sbin
LIBEXEC         = $(RELEASE_DIR)/libexec
HISTORY         = $(SPOOL)/history
##  Where is the Stork binary installed?
STORK                       = $(SBIN)/stork_server
STORK_ADDRESS_FILE = $(LOG)/.stork_address
## $(STORK_LOG_BASE): Stork server job queue classad collection journal file.
## $(STORK_LOG_BASE).history: Used to track completed jobs.
## $(STORK_LOG_BASE).user_log: User level log, also used by DAGMan.

STORK_LOG_BASE          = $(LOG)/Stork
STORK_LOG = $(LOG)/StorkLog
STORK_DEBUG = D_FULLDEBUG
MAX_STORK_LOG = 4000000

## Stork startup arguments ## Start Stork on a well-known port.
STORK_PORT                      = 9621
STORK_ARGS = -p $(STORK_PORT) -f -Serverlog $(STORK_LOG_BASE)

## Limits the number of retries for a failed data placement (default= 10)
```

```
STORK_MAX_RETRY = 1
## Limits the run time for a data placement job,
after which the placement is considered failed. (by default = 0 - infinite)
#STORK_MAXDELAY_INMINUTES = 10
## Temporary credential storage directory used by Stork.
#STORK_TMP_CRED_DIR = /tmp
## Directory containing Stork modules.
STORK_MODULE_DIR = $(LIBEXEC)
## max number of jobs running at the same time (default 1)
STORK_MAX_NUM_JOBS = 1
## Temporary credential storage directory used by Stork.
##STORK_TMP_CRED_DIR = /tmp
## aggregating jobs - (default is 0, no aggregation)
## 1: aggregation according to dest_url
## 2:                          src_url
## 3: if src_hostname and dest_hostname matches
## 4:                        dest_hostname
##                          src_hostname
#STORK_AGGR_LEVEL  = 0
## set max number of job that can be aggregated (combined into a single job)
#STORK_AGGR_MAX_COUNT = 0
## @@  transfer module specific default values @@ ##
##
# STORK_RECURSIVE_COPY = FALSE
##
# STORK_VERIFY_CHECKSUM = FALSE
##
# STORK_VERIFY_FILESIZE = FALSE
##
#STORK_NETWORK_CHECK = FALSE
##
# STORK_TRANSFER_CHECKPOINT = FALSE
##
# STORK_TRANSFER_CHECKPOINT = FALSE
##
# STORK_SYNC_ONLY = FALSE
##
# STORK_SET_PERMISSIONS = FALSE
##
# STORK_TEST_MODE=0
```

3 Running Stork

If installation is made with root privileges, the installation script will try to create a *stork* user. The server will switch to *stork* user for security purposes.

The command below starts the stork_server connected to port 10000. The stork logs are named with the prefix Stork, such as StorkLog, Stork.history, etc.

```
./stork_server -p <port> -Serverlog <stork/log/directory/prefixForLogFiles>
```

The Stork server generates a log file which is used for logging the activities of the Stork server.

Below is a sample of the log file:

```
cat local/log/StorkLog
0/23 16:55:24 ******************************************************
10/23 16:55:24 ** stork_server (STORK) STARTING UP
10/23 16:55:24 ** /tmp/sbin/stork_server
10/23 16:55:24 ** $CondorVersion: 6.9.4 Sep 11 2008 $
10/23 16:55:24 ** $CondorPlatform: I386-LINUX_RHEL3 $
10/23 16:55:24 ** PID = 13697
10/23 16:55:24 ** Log last touched time unavailable (No such file or directory)
10/23 16:55:24 ******************************************************
10/23 16:55:24 Using config source: /tmp/etc/stork_config
10/23 16:55:24 DaemonCore: Command Socket at <208.100.92.21:47661>
10/23 16:58:21 ******************************************************
10/23 16:58:21 ** stork_server (STORK) STARTING UP
10/23 16:58:21 ** /tmp/sbin/stork_server
10/23 16:58:21 ** $Version: 6.9.4 Sep 11 2008 $
10/23 16:58:21 ** $Platform: I386-LINUX_RHEL3 $
10/23 16:58:21 ** PID = 14336
10/23 16:58:21 ** Log last touched 10/23 16:55:24
10/23 16:58:21 ******************************************************
10/23 16:58:21 Using config source: /tmp/etc/stork_config
10/23 16:58:21 DaemonCore: Command Socket at <208.100.92.21:47712>
10/23 16:58:21 ============================================================
10/23 16:58:21               STORK CONFIGURATION:
10/23 16:58:21 ============================================================
10/23 16:58:21 DaP log file     : storkserver.log
10/23 16:58:21 Userlog file     : (null)
10/23 16:58:21 XML log file     : (null)
10/23 16:58:21 Client Agent host: (null)
10/23 16:58:21 ============================================================
10/23 16:58:21 STORK_TEST_MODE = 10 (0: do testing only - 10: run as server)
10/23 16:58:21 STORK_MAX_NUM_JOBS = 1
10/23 16:58:21 STORK_MAX_RETRY = 1
10/23 16:58:21 STORK_MAXDELAY_INMINUTES = 0
10/23 16:58:21 STORK_AGGR_LEVEL = 0 - 0: no aggregation
 - 1: according to dest url; 2: according to src url - 3: if hostnames match
10/23 16:58:21 STORK_MAX_COUNT = 1 -
10/23 16:58:21 STORK_RECURSIVE_COPY = FALSE
10/23 16:58:21 STORK_VERIFY_CHECKSUM = FALSE
10/23 16:58:21 STORK_VERIFY_FILESIZE = FALSE
10/23 16:58:21 STORK_NETWORK_CHECK = FALSE
10/23 16:58:21 STORK_TRANSFER_CHECKPOINT = FALSE
10/23 16:58:21 STORK_SYNC_ONLY = FALSE
10/23 16:58:21 STORK_TMP_CRED_DIR = /tmp
10/23 16:58:21 STORK_MODULE_DIR = /tmp/libexec
10/23 16:58:21 modules will execute in LOG directory /tmp/local/log
10/23 16:58:21 Getting monitoring info for pid 14336
.....................
.....................
```

4 Stork Components

stork_server:

The *stork_server* is the main component of the Stork scheduler. The stork_server runs as a persistent daemon process and performs all management, scheduling, execution, and monitoring of data placement activities.

The Stork server accepts the following parameters (defined by STORK_ARGS)

```
$ sbin/stork_server --help
============================================================
USAGE: stork_server
          [ -t           ] // output to stdin
          [ -p           ] // port on which to run Stork Server
          [ -help        ] // stork help screen
          [ -Config      ] // stork config file
          [ -Serverlog   ] // stork server log in ClassAds
          [ -Xmllog      ] // stork server log in XML format
          [ -Userlog     ] // stork userlog in XMLformat
          [ -Clientagent ] // host where client agent is running
============================================================
```

stork_submit:

The *stork_submit* is a client side tool used to submit stork jobs to the stork_server.

```
$ bin/stork_submit
usage: stork_submit  [option]... [stork_server] submit_file
stork_server                    specify explicit stork server (deprecated)
submit_file                     stork submit file
        -lognotes "notes"       add lognote to submit file before processing
        -stdin                  read submission from stdin instead of a file
        -help                   print this help information
        -version                print version information
        -debug                  print debugging information to console
        -name stork_server      stork server
```

stork_status:

The *stork_status* is a client side tool used to query regarding the status of jobs submitted to the stork_server.

The *dap_id* is used by the stork_status commmand to query the Stork server. The dap_id is generated and assisgned to a job when it is submitted to Stork using the stork_submit command.

The stork_status command accepts the following parameters, where host_name is optional. The host_name is used to specify a Stork server on a remote host.

```
 ./stork_status -h
usage: stork_status  [option]... [stork_server] job_id
stork_server                    specify explicit stork server (deprecated)
job_id                          stork job id
        -help                   print this help information
        -version                print version information
        -debug                  print debugging information to console
        -name stork_server      stork server
```

stork_rm:

The *stork_rm* is a client side tool used to delete any jobs that are currently queued with the stork_server.

```
./stork_rm -h
usage: stork_rm  [option]... [stork_server] job_id
stork_server                    specify explicit stork server (deprecated)
job_id                          stork job id
        -help                   print this help information
        -version                print version information
        -debug                  print debugging information to console
        -name stork_server      stork server
```

stork_q:

The *stork_q* is a client side tool used to retrieve a listing of jobs that are currently queued with the stork_server.

```
 ./stork_q -h
usage: stork_q  [option]... [stork_server]
stork_server                    stork server (deprecated)
        -help                   print this help information
        -version                print version information
        -debug                  print debugging information to console
        -name stork_server      stork server
```

Sample output from the stork_q command:

```
[
    dest_url = "file:///home/user1/stork/data10M_48";
    src_url = "file:///home/user1/stork/data10M";
    remote_user = "user1@dsl-turtle06.cct.lsu.edu";
    status = "request_rescheduled";
    dap_id = 264;
    use_protocol = 0;
    stork_server = "qb1.loni.org";
    dap_type = "transfer";
    error_code = "port not accessible";
    num_attempts = 1;
    owner = "user1";
    cluster_id = 264;
    timestamp = absTime("2008-05-28T14:52:22-0500");
    generic_event = "Rescheduling."
]
```

Stork server runs as a persistent daemon process. It consistently listens to requests from the clients. The clients send their requests to the Stork server over the network using stork_submit command line tool in form of a ClassAd (Classified Advertisement).

Since Stork is designed to work on a heterogeneous computing environment,one of its goals is to support as many storage systems and file transfer protocols as possible.

Another important characteristic of Stork is reliability. It makes sure that the requested transfers are completed successfully even in case of server or network failures.

Stork source and destination URLs have a naming convention. All URLs ending with a slash (/) are assumed to be directories and the rest are assumed to be files.

5 New Features

Here is a possible submit file including extended features.

```
 etc/submitArgument.sample
[
        dap_type = "transfer";
        src_url  = "gsiftp://$src.loni.org/home/balman/tests/$srcfile";
        dest_url = "gsiftp://$dest.loni.org/home/balman/tests/dest-$destfile";
        output   =  "out";
        err      =  "err";
        arguments = "-p 10";
        set_permission          = "066";
        sync_only               = true;
        checkpoint_transfer     = true;
        network_check           = true;
        verify_filesize         = true;
        recursive_copy          = true;
]
```

File size Verification Support

Currently all the transfer modules supported by Stork support file size verification. File size verification can either be turned ON/OFF by specifying the corresponding option in the Stork configuration file.

When switched ON, Stork determines the filesizes of the files at the source and the files at the destination and compares them. If the filesizes differ, an error message is logged in the Stork log file.

Checksum Verification Support

Currently all the transfer modules supported by Stork support checksum verification. Checksum verification can either be turned ON/OFF by specifying the corresponding option in the Stork configuration file.

When switched ON, Stork computes the checksums of the files at the source and the files at the destination and compares them. If the checksums differ, an error message is logged in the Stork log file.

Recursive Transfers

Currently all the transfer modules supported by Stork support recursive directory transfers. Recursive directory transfers are specified in the URL by ending the URLs with a '/' to represent a directory.

Wild Card Support

Currently all the transfer modules supported by Stork support transferring files with a wild cards such as *.txt, stork*, *stork or st*rk.

Checkpointing File Transfers

Currently the Petashare and GridFTP transfer modules supported by Stork support checkpointing of transfer and provide the capability of resuming transfers in the event of an error.

These Stork modules checkpoint the transfers during various stages and thus enable Stork to resume the transfer at the last checkpoint in the event of a network outage or crash.

6 Job Submission

Since Stork is designed to work on a heterogeneous computing environment,one of its goals is to support as many storage systems and file transfer protocols as possible. Currently the following protocols and storage systems are supported by Stork:

- file
- FTP
- GridFTP
- HTTP

- iRODS
- PetaShare
- SRB

The protocol to be used is determined by the Stork server according to the URL signatures of the files to be transfered.

URLs supported: The format of the URL for various supported protocols is as below:

- file URL - file:///path/to/file
- FTP URL - ftp://user:password@host:port/path/to/file
- HTTP URL - http://user:password@host:port/path/to/file
- GridFTP URL - gsiftp://user:password@host:port/path/to/file
- SRB URL - srb://user[.mdasDomain[.zone]]:password@host:port/path/to/file
- iRODS URL - irods://user.zone:password@host:port/path/to/file
- PetaShare - petashare://path/to/file

Assuming that a service (SRB, iRODS, GridFTP, etc) is running: Please note that in the URLs shown above the parameters denote:

- user - the username
- password - password corresponding to the username
- host - the host on which the service is running
- port - the port the service listens on
- /path/to/file - path to the location of the file you would like to transfer
- /path/to/directory/ - path to the directory you would like to perform transfers

Sample Stork Job Requests

i) file to file transfer

```
[
  dap_type = "transfer";
  src_url = "file:///path/to/file";
  dest_url = "irods://user.zone:password@host:port/path/to/file";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/file";
  dest_url = "file:///path/to/file";
]
```

ii) file selection using wild cards

```
[
  dap_type = "transfer";
  src_url = "file:///path/to/file*";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:///path/to/*file";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:///path/to/fi*le";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:///path/to/*";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/file*";
  dest_url = "file:///path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/*file";
  dest_url = "file:///path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/fi*le";
  dest_url = "file:///path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/*";
  dest_url = "file:///path/to/directory/";
]
```

iii) recursive transfer from local directory to SRB collection

```
[
  dap_type = "transfer";
  src_url = "file:///path/to/directory/";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/directory/";
  dest_url = "file:///path/to/directory/";
]
```

Any of the supported protocols may be invoked by simply replacing the URLs shown above by those of the protocols required. The new URLs should however conform to their URL format as described in the supported URL format section above.